A Cloning Pushout Approach to Term-Graph Transformation

D. Duval[1], R. Echahed[2], and F. Prost[2]

[1] Laboratoire LJK
B. P. 53, F-38041 Grenoble, France
Dominique.Duval@imag.fr
[2] Laboratoire LIG
46, av Félix Viallet, F-38031 Grenoble, France
Rachid.Echahed@imag.fr/ Frederic.Prost@imag.fr

Abstract. We address the problem of cyclic termgraph rewriting. We propose a new framework where rewrite rules are tuples of the form (L, R, τ, σ) such that L and R are termgraphs representing the left-hand and the right-hand sides of the rule, τ is a mapping from the nodes of L to those of R and σ is a partial function from nodes of R to nodes of L. τ describes how incident edges of the nodes in L are connected in R. τ is not required to be a graph morphism as in classical algebraic approaches of graph transformation. The role of σ is to indicate the parts of L to be cloned (copied). Furthermore, we introduce a new notion of *cloning pushout* and define rewrite steps as cloning pushouts in a given category. Among the features of the proposed rewrite systems, we quote the ability to perform local and global redirection of pointers, addition and deletion of nodes as well as cloning and collapsing substructures.

1 Introduction

Complex data-structures built by means of records and pointers, can formally be represented by *termgraphs* [2, 14, 11]. Roughly speaking, a termgraph is a first-order term with possible sharing and cycles. The unravelling of a termgraph is a rational term. Termgraph rewrite systems constitute a high-level framework which allows one to describe, at a very abstract level, algorithms over data-structures with pointers. Thus avoiding, on the one hand, the cumbersome encodings which are needed to translate graphs (data-structures) into trees in the case of programing with first-order term rewrite systems and, on the other hand, the many classical errors which may occur in imperative languages when programing with pointers.

Transforming a termgraph is not an easy task in general. Many different approaches have been proposed in the literature which tackle the problem of termgraph transformation. The algorithmic approach such as [2] defines in details every step involved in the transformation of a term-graph by providing the corresponding algorithm. This approach is too close to implementation techniques.

In [1], equational definition of term-graphs are exploited to define termgraph transformation. These transformations are obtained up to bisimilar structures (two termgraphs are bisimilar if they represent the same rational term). Unfortunately, bisimilarity is not a congruence in general (e.g., the lengths of two bisimilar but different circular lists are not bisimilar).

A more abstract approach to graph transformation is the algebraic one, first proposed in the seminal paper [7]. It defines a rewrite step using the notion of pushouts. The algebraic approach is quite declarative. The details of graph transformations are hidden thanks to pushout constructs. There are mainly two different algebraic approaches, namely the double pushout (DPO) and the single pushout (SPO) approaches.

In the DPO approach [7, 4], a rule is defined as a pair of graph morphisms $L \leftarrow K \rightarrow R$ where L, K and R are graphs and the arrows represent graph homomorphisms. A graph G rewrites into a graph H, iff there exists a homomorphism (a matching) $m : L \rightarrow G$ and a graph D such that the left and the right squares of the diagram of Fig.1 are pushouts.

$$\begin{tikzcd}
L \arrow[d, "m"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "d"] & R \arrow[d, "m'"] \\
G & D \arrow[l, "l'"] \arrow[r, "r'"'] & H
\end{tikzcd}$$

Fig. 1. Double pushout: a rewrite step

$$\begin{tikzcd}
L \arrow[r, "l"'] \arrow[d, "m"'] & R \arrow[d, "m'"] \\
G \arrow[r, "l'"'] & H
\end{tikzcd}$$

Fig. 2. Single pushout: a rewrite step

In general, D is not unique. Sufficient conditions may be given such as dangling and identification conditions in order to ensure existence of pushout complement. The DPO approach is easy to grasp since morphisms are supposed to be completely defined. However, this approach fails, in general, to specify rules with deletion of nodes. For example, if we consider the rule $f(x) \rightarrow f(b)$ which can be translated into the span $f(x) \leftarrow K_0 \rightarrow f(b)$ for some graph K_0, and apply that rule on $f(a)$, then because of pushout properties $f(a)$ is rewritten into a termgraph H which contains a. However, $f(b)$ is the only desired result for H.

In the SPO approach [13, 8, 9, 6], a rule is a *partial* graph morphism $L \rightarrow R$. When a (total) graph morphism $m : L \rightarrow G$ exists, G can rewrite to a graph H iff the square of Fig 2 is a pushout. This approach is appropriate to specify deletion of nodes thanks to partial homomorphisms. However, in the case of termgraphs, some care should be taken when a node is deleted. Indeed, deletion of a node causes automatically the deletion of its incident edges. This is not sound in the case of termgraphs since each function symbol should have as many successors as its arity.

In this paper, we investigate a new approach to the definition of rewrite relations over cyclic termgraphs. We are interested in rewrite relations, R, over termgraphs such that (t, t') belongs to R, iff t' is obtained from t by performing a series of actions of the six following kinds :(i) addition of new nodes, (ii) redirection of particular edges, (iii) redirection of all incident edges of a particular node (iv) deletion of nodes (v) cloning of nodes and (vi) collapsing of nodes. In order to deal with these features in a single framework, we propose a new algebraic approach to define such rewrite relations. Our approach departs from the SPO and the DPO approaches. A rewrite rule is defined as a tuple (L, R, τ, σ) such that, L and R are termgraphs, respectively the left-hand side and the right-hand side of the rule. τ is a mapping from the nodes of L into the nodes of R (τ has not to be a graph morphism). $\tau(n) = n'$ indicates that incident edges of n are to be redirected towards n'. σ is a partial function from unlabeled nodes of R into nodes of L. Roughly speaking, $\sigma(n) = p$ indicates that node n should be instantiated as p (parameter passing). We show that whenever a matching $m : L \to G$ exists, then the termgraph G rewrites into a termgraph H. We define the termgraph H as an initial object of a given category. The construction of H could be seen as a generalization of that of pushouts. We call it *cloning pushout*.

The paper is organized as follows. In the next section we introduce the basic definitions of graphs and morphisms that we consider in the paper. In section 3, we introduce a first simplified version of our rewriting approach. This first step prevents from the cloning of substructures. Then, in section 4, we give the full definition of rewriting, including cloning possibility, and illustrate our approach through several examples in section 5. Concluding remarks are given in section 6.

2 Graphs

In this section we give some technical definitions that we use in the paper. We assume the reader is familiar with category theory. The missing definitions may be consulted in [10].

Throughout this paper, a signature Ω is fixed. Each operation symbol $\omega \in \Omega$ is endowed with an *arity* $\mathrm{ar}(\omega) \in \mathbb{N}$. For each set X, the set of strings over X is denoted X^*, and for each function $f : X \to Y$, the function $f^* : X^* \to Y^*$ is defined by $f^*(x_1 \dots x_n) = f(x_1) \dots f(x_n)$.

Definition 21 (Graph) A *termgraph*, or simply a *graph* $G = (\mathcal{N}, \mathcal{D}, \mathcal{L}, \mathcal{S})$ is made of a set of *nodes* $\mathcal{N}$ and a subset of *labeled nodes* $\mathcal{D} \subseteq \mathcal{N}$, which is the domain for a *labeling function* $\mathcal{L} : \mathcal{D} \to \Omega$ and a *successor function* $\mathcal{S} : \mathcal{D} \to \mathcal{N}^*$, such that for each labeled node n, the length of the string $\mathcal{S}(n)$ is the arity of the operation $\mathcal{L}(n)$. For each labeled node n the fact that $\omega = \mathcal{L}(n)$ is written $n{:}\omega$, and each unlabeled node n may be written as $n{:}\bullet$, so that the symbol $\bullet$ is a kind of anonymous variable.

A *graph homomorphism*, or simply a *graph morphism* $g : G \to H$, where $G = (\mathcal{N}_G, \mathcal{D}_G, \mathcal{L}_G, \mathcal{S}_G)$ and $H = (\mathcal{N}_H, \mathcal{D}_H, \mathcal{L}_H, \mathcal{S}_H)$ are graphs, is a function

$g : \mathcal{N}_G \to \mathcal{N}_H$ which preserves the labeled nodes and the labeling and successor functions. This means that $g(\mathcal{D}_G) \subseteq \mathcal{D}_H$, and for each labeled node n, $\mathcal{L}_H(g(n)) = \mathcal{L}_G(n)$ and $\mathcal{S}_H(g(n)) = g^*(\mathcal{S}_G(n))$ (the image of an unlabeled node may be any node). This yields the category **Gr** of graphs.

We denote by **Set** the classical category of sets.

Definition 22 (Node functor) The *node functor* $|-| : \mathbf{Gr} \to \mathbf{Set}$ maps each graph $G = (\mathcal{N}, \mathcal{D}, \mathcal{L}, \mathcal{S})$ to its set of nodes $|G| = \mathcal{N}$ and each graph morphism $g : G \to H$ to its underlying function $|g| : |G| \to |H|$.

We may denote g instead of $|g|$ since the node functor is faithful, which means that a graph morphism is determined by its underlying function on nodes. The faithfulness of the node functor implies that a diagram of graphs is commutative if and only if its image by the node functor is commutative, as a diagram of sets. It may be noted that the node functor preserves pullbacks, because it has a left adjoint, and that it does not preserve pushouts.

The following definition introduces a new notion of *graphic functions.* These functions are used to relate graphs involved in a rewrite step, in addition to classical graph homomorphisms.

Definition 23 (Graphic functions) Let G and H be graphs and $\gamma : |G| \to |H|$ a function. For each node n of G, γ is *graphic at n* if either n is unlabeled or both n and $\gamma(n)$ are labeled, $\mathcal{L}_H(\gamma(n)) = \mathcal{L}_G(n)$ and $\mathcal{S}_H(\gamma(n)) = \gamma^*(\mathcal{S}_G(n))$. And γ is *strictly graphic at n* if either both n and $\gamma(n)$ are unlabeled or both n and $\gamma(n)$ are labeled, $\mathcal{L}_H(\gamma(n)) = \mathcal{L}_G(n)$ and $\mathcal{S}_H(\gamma(n)) = \gamma^*(\mathcal{S}_G(n))$. For each set of nodes Γ of G, γ is *graphic (resp. strictly graphic) on Γ* if γ is graphic (resp. strictly graphic) at every node in Γ.

It should be noted that the property of being graphic (resp. strictly graphic) on Γ involves the successors of the nodes in Γ, which may be outside Γ.

Example 1. Let us consider the graphs $G1$ and $G2$ given respectively in Fig 3 and Fig 4. Let $\Gamma_1 = \{1, 3\}$, $\Gamma_2 = \{1, 2, 3\}$ and $\Gamma_3 = \{1, 2, 3, 4\}$. Let $\gamma : |G| \to |H|$ be the function defined by $\gamma = \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c, 4 \mapsto d\}$. It is easy to check that γ is graphic on Γ_2, γ is strictly graphic on Γ_1, γ is not strictly graphic on Γ_2 and γ is not graphic on Γ_3.

$$\begin{tikzcd}
& 1 : f \arrow[dl] \arrow[d] \arrow[dr] & \\
2 : \bullet & 3 : \bullet & 4 : nil
\end{tikzcd}$$

Fig. 3. G1

$$\begin{tikzcd}
& a : f \arrow[dl] \arrow[d] \arrow[dr] & \\
b : nil & c : \bullet & d : \bullet
\end{tikzcd}$$

Fig. 4. G2

Clearly, a function $\gamma : |G| \to |H|$ underlies a graph morphism $g : G \to H$ if and only if it is graphic on $|G|$. The next straightforward result will be useful.

Lemma 24 *Let G, H, H' be graphs and let $\gamma : |G| \to |H|$, $\gamma' : |G| \to |H'|$, $\eta : |H| \to |H'|$ be functions such that $\gamma' = \eta \circ \gamma$. Let Γ be a set of nodes of G. If γ is strictly graphic on Γ and γ' is graphic on Γ, then η is graphic on $\gamma(\Gamma)$.*

3 Rewriting without cloning

Roughly speaking, in the context of graph rewriting, a rewrite rule has a left-hand side graph L and a right-hand side graph R, and a rewrite step applied to a graph G with an occurrence of L returns a graph H with an occurrence of R, by replacing L by R in G. We deal with termgraphs, so that a labeled node p in G outside L and with its i-th successor p' in L must have some i-th successor n' in H. For this purpose, we introduce a "target" function τ, from the nodes of L to the nodes of R, and we decide that n' must be $\tau(p')$. The aim of this section is to define this process precisely. The definitions and results in this section are simplified versions of those in the next section.

In this section, a *rewrite rule* is tuple (L, R, τ) made of two graphs L and R and a (total) function $\tau : |L| \to |R|$. A *morphism of rewrite rules* from $T = (L, R, \tau)$ to $T_1 = (L_1, R_1, \tau_1)$ is a pair of graph morphisms (m, d) with $m : L \to L_1$, $d : R \to R_1$ such that $|d| \circ \tau = \tau_1 \circ |m|$.

In this paper, the illustrations take place either in the category **Set** of sets or in a heterogeneous framework where the points stand for graphs, the solid arrows for graph morphisms and the dashed arrows for functions on nodes. So, a rewrite rule $T = (L, R, \tau)$ can be illustrated as follows:

$$\begin{tikzcd} L \arrow[r, dashed, "\tau"'] & R \end{tikzcd}$$

It can be noted that each graph morphism $t : L \to R$ determines a rewrite rule where $\tau = |t|$. In this case, for each graph morphism $m : L \to G$ the pushout of t and m in the category **Gr** is defined as the initial object in the category of cones over t and m. Let us generalize this definition to any rewrite rule $T = (L, R, \tau)$ and any graph morphism $m : L \to G$. A *heterogeneous cone over T and m* is made of a graph H, a function $\tau_1 : |G| \to |H|$ and a graph morphism $d : R \to H$ such that $T_1 = (G, H, \tau_1)$ is a rewrite rule, $(m, d) : T \to T_1$ is a morphism of rewrite rules and τ_1 is graphic on $|G| - |m(L)|$.

$$\begin{tikzcd} L \arrow[r, dashed, "\tau"'] \arrow[d, "m"'] & R \arrow[d, "d"] \\ G \arrow[r, dashed, "\tau_1"'] & H \end{tikzcd}$$

A *morphism of heterogeneous cones over T and m*, say $h : (H, \tau_1, d) \to (H', \tau_1', d')$, is a graph morphism $h : H \to H'$ such that $|h| \circ \tau_1 = \tau_1'$ and $h \circ d = d'$. This yields the category $\mathbf{H}_{T,m}$ of heterogeneous cones over T and m. A *heterogeneous pushout* of T and m is defined as an initial object in the category $\mathbf{H}_{T,m}$.

When a heterogeneous pushout exists, its initiality property implies that it is unique up to an isomorphism of heterogeneous cones. A *matching* of a graph L is a graph morphism $m : L \to G$ such that $|m|$ is injective. It is easy to prove the existence of a heterogeneous pushout of a rewrite rule $T = (L, R, \tau)$ and a matching $m : L \to G$, as follows. Let $(\mathcal{P})$ denote the following pushout of τ and $|m|$ in **Set**:

$$\begin{tikzcd}
|L| \arrow[r, "\tau"'] \arrow[d, "|m|"'] & |R| \arrow[d, "\delta"] \\
|G| \arrow[r, "\tau_1"'] & \mathcal{H}
\end{tikzcd}$$

Then $\mathcal{H} = \tau_1(|G| - |m(L)|) + \delta(|R|)$, and in addition the restriction of $\tau_1 : |G| - |m(L)| \to \tau_1(|G| - |m(L)|)$ is bijective and the restriction of $\delta : |R| \to \delta(|R|)$ is bijective. Hence, a graph H with set of nodes $\mathcal{H}$ is defined simply by imposing that τ_1 is strictly graphic on $|G| - |m(L)|$ and that δ is strictly graphic on $|R|$. It follows that $\delta = |d|$ for a graph morphism $d : R \to H$ and that (H, τ_1, d) forms a heterogeneous cone over T and m. Now, let us consider any heterogeneous cone (H', τ_1', d') over T and m. Because of the pushout of sets $(\mathcal{P})$, there is a unique function $\eta : |H| \to |H'|$ such that $\eta \circ \tau_1 = \tau_1'$ and $\eta \circ |d| = |d'|$. In addition, it follows from lemma 24 that η is graphic on $\tau_1(\Gamma)$ and also on $d(|R|)$. So, η underlies a graph morphism $h : H \to H'$. Since the node functor is faithful, it follows that (H, τ_1, d) is a heterogeneous pushout of T and m.

Now, given a rewrite rule $T = (L, R, \tau)$ and a matching $m : L \to G$, the corresponding *rewrite step* builds the graph morphism $d : R \to H$, obtained from the heterogeneous pushout of T and m. It can be noted that d is a matching of R.

The induced rewrite relation over termgraphs is unfortunately not satisfactory. Consider for instance the rule $f(x) \to g(x, x)$. Informally, the application of such a rule on the termgraph $1 : f(2 : a)$ can yield either the termgraph $1 : g(2 : a, 2)$ or the termgraph $1 : g(2 : a, 3 : a)$ according to the way the term $g(x, x)$ is represented as a termgraph. However, the application of the definition of a rewrite step, as given above, suggests to rewrite the termgraph $1 : f(2 : a)$ into $1 : g(2 : \bullet, 2)$ by means of the following rule $(1 : f(x : \bullet), 1 : g(x : \bullet, x), \tau = \{1 \mapsto 1, x \mapsto x\})$. The node 2 is not labeled in the reduced termgraph. This reflects the fact that the instance of x cannot be substituted or cloned in the right-hand side. We overcome this drawback in the next section.

4 Rewriting with cloning

In this section, the definitions and results of the previous section are generalized in order to add a "cloning" process. Indeed, in the resulting graph H from section 3 there is no node in R with its image outside R. This is an issue, which is solved in this section thanks to the notion of "clone". Roughly speaking, a clone of a labeled node p in G is a node n in H with the same label and "the same" successors as p, where "the same" successors are defined via the target function τ from the previous section. The definition of a rewrite rule is generalized so that it yields the information about the way the images of the nodes in L must be cloned by images of nodes in R. The main result is theorem 48: under relevant definitions and assumptions, for each rewrite rule T and matching m there is a *cloning pushout* of T and m, which can be built explicitly from a pushout of sets. Since each node in L may have an arbitrary number of clones (maybe no clone at all), and a node in R cannot be a clone of more than one node in L, the relation between the nodes in L and their clones in R takes the form of a partial function from $|R|$ to $|L|$. In this paper, partial functions are denoted with the symbol "$\rightharpoonup$", the domain of a partial function σ is denoted $\mathrm{Dom}(\sigma)$, and the composition of partial functions is defined as usual.

Definition 41 (Clones) Let G and H be graphs and $\tau : |G| \to |H|$ a function. Then $p \in |H|$ is a τ*-clone* of $q \in |G|$ when: p is labeled if and only if q is labeled, and then $\mathcal{L}_H(p) = \mathcal{L}_G(q)$ and $\mathcal{S}_H(p) = \tau^*(\mathcal{S}_G(q))$.

Definition 42 (Rewrite rule) A *rewrite rule* is tuple (L, R, τ, σ) made of two graphs L and R, a function $\tau : |L| \to |R|$ and a partial function $\sigma : |R| \rightharpoonup |L|$ such that each node n in the domain of σ is unlabeled or is a τ-clone of $\sigma(n)$. A *morphism of rewrite rules*, from $T = (L, R, \tau, \sigma)$ to $T_1 = (L_1, R_1, \tau_1, \sigma_1)$ is a pair of graph morphisms (m, d) with $m : L \to L_1$ and $d : R \to R_1$ such that $|d| \circ \tau = \tau_1 \circ |m|$, $d(\mathrm{Dom}(\sigma)) \subseteq \mathrm{Dom}(\sigma_1)$ and $|m| \circ \sigma = \sigma_1 \circ |d|$ on $\mathrm{Dom}(\sigma)$.

In the previous section, we have dealt with the simple case where the domain of σ is empty.

In the sequel, a rewrite rule $T = (L, R, \tau, \sigma)$ will be illustrated as follows:

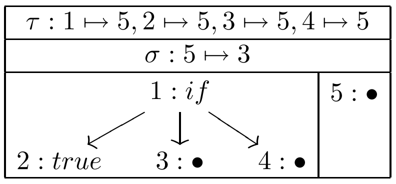
$$\begin{tikzcd} L \arrow[r, dashed, "\tau"'] & R \arrow[l, dashed, bend right, "\sigma"'] \end{tikzcd}$$

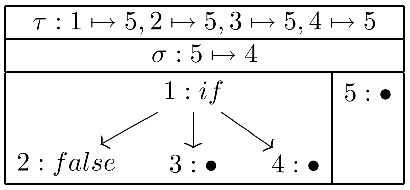
or depicted as opposite, where the lines τ and σ contain the definitions of the functions τ and σ.

τ:	
σ:	
L	R

Example 2 (if-then-else).

Below, we give the rewrite rules which define the If-then-else operator as it behaves in classical imperative languages.





The definition of τ ensures that the if-then-else expression is replaced by its value $\tau(1) = 5$. The definition of σ indicates that the value of the if-then-else is its second (resp. third) argument specified by $\sigma(5) = 3$ (resp. $\sigma(5) = 4$) in the rules above. Notice that if σ were defined as the empty function, the if-then-else expression would evaluate to an unlabeled node.

Example 3 (Cloning data-structures). In this example we give the rules to clone natural numbers, encoded with *succ* and *zero*. The clone of *zero* is done using the following rule:

$$\tau : 1 \mapsto 2, 2 \mapsto 2$$
$$\sigma : 3 \mapsto 2, 2 \mapsto 2$$
$$\begin{tikzcd}
1 : \mathit{clone} \arrow[d] & \arrow[l, phantom, "\mid"] 2 : \mathit{zero} & 3 : \mathit{zero} \\
2 : \mathit{zero} & &
\end{tikzcd}$$

One can note that the condition on the labeled nodes (ie τ-clones, see def. 42) in the domain of σ is verified. This rule redirects all edges from 1 to 2, while the edges adjacent to 2 remain unchanged.

The second rule is defined as follows:

$$\tau : 1 \mapsto 4, 2 \mapsto 2, 3 \mapsto 3$$
$$\sigma : 3 \mapsto 3$$
$$\begin{tikzcd}
1 : \mathit{clone} \arrow[r] & 2 : \mathit{succ} \arrow[d] & \arrow[l, phantom, "\mid"] 2 : \mathit{succ} \arrow[d] & 4 : \mathit{succ} \arrow[d] \\
 & 3 : \bullet & 3 : \bullet & 5 : \mathit{clone} \arrow[l]
\end{tikzcd}$$

Notice that, in this case, it is not possible to define $\sigma(4) = 2$ because sucessor of 4 in R is labeled by *clone* and successor of 2 in L is labeled by *succ*, thus breaking the τ-clone condition.

Definition 43 (Cloning cone) Let $T = (L, R, \tau, \sigma)$ be a rewrite rule and $m : L \to G$ a graph morphism. A *cloning cone over* T *and* m is a tuple (H, τ_1, d, σ_1) made of a graph H, a function $\tau_1 : |G| \to |H|$, a graph morphism $d : R \to H$ and a partial function $\sigma_1 : |H| \rightharpoonup |G|$ such that $T_1 = (G, H, \tau_1, \sigma_1)$ is a rewrite rule, $(m, d) : T \to T_1$ is a morphism of rewrite rules, τ_1 is graphic on $|G| - |m(L)|$ and n_1 is a τ_1-clone of $\sigma_1(n_1)$ for each n_1 in the domain of σ_1.

$$\begin{tikzcd}
L \arrow[r, dashed, "\tau"'] \arrow[d, "m"'] & R \arrow[l, dashed, bend right, "\sigma"', harpoon] \arrow[d, "d"] \\
G \arrow[r, dashed, "\tau_1"'] & H \arrow[l, dashed, bend right, "\sigma_1"', harpoon]
\end{tikzcd}$$

A *morphism of cloning cones over* T *and* m, say $h : (H, \tau_1, d, \sigma_1) \to (H', \tau_1', d', \sigma_1')$, is a graph morphism $h : H \to H'$ such that $|h| \circ \tau_1 = \tau_1'$, $h \circ d = d'$, $h(\mathrm{Dom}(\sigma_1)) \subseteq$

Dom(σ_1') and $\sigma_1' \circ |h| = \sigma_1$ on Dom(σ_1).
This yields the category $\mathbf{C}_{T,m}$ of cloning cones over T and m.

Definition 44 (Cloning pushout) Let $T = (L, R, \tau, \sigma)$ be a rewrite rule and $m : L \to G$ a graph morphism. A *cloning pushout of* T *and* m is an initial object in the category $\mathbf{C}_{T,m}$ of cloning cones over T and m.

When a cloning pushout exists, its initiality implies that it is unique up to an isomorphism of cloning cones. In theorem 48 we prove the existence of a cloning pushout of T and m under some injectivity assumption on m.

Definition 45 (Matching) A *matching* with respect to a rewrite rule $T = (L, R, \tau, \sigma)$ is a graph morphism $m : L \to G$ such that if $m(p) = m(p')$ for distinct nodes p and p' in L then $\tau(p)$ and $\tau(p')$ are in Dom(σ) and $\sigma(\tau(p)) = \sigma(\tau(p'))$ in L.

Proposition 46 *Let $T = (L, R, \tau, \sigma)$ be a rewrite rule and $m : L \to G$ a matching with respect to T. Then the pushout of τ and $|m|$ in* **Set**:

$$\begin{tikzcd}
|L| \arrow[r, "\tau"'] \arrow[d, "|m|"'] & |R| \arrow[d, "\delta"] \\
|G| \arrow[r, "\tau_1"'] & \mathcal{H}
\end{tikzcd}$$

satisfies:

$$\mathcal{H} = \tau_1(\Gamma) + \delta(\Delta) + \delta(\Sigma)$$

where $\Gamma = |G| - |m(L)|$, $\Sigma = \mathrm{Dom}(\sigma)$, $\Delta = |R| - \Sigma$ and:

- *the restriction of $\tau_1 : \Gamma \to \tau_1(\Gamma)$ is bijective,*
- *the restriction of $\delta : \Delta \to \delta(\Delta)$ is bijective,*
- *and the restriction of $\delta : \Sigma \to \delta(\Sigma)$ is such that if $\delta(n) = \delta(n')$ for distinct nodes n and n' in Σ then $\sigma(n) = \sigma(n')$ in L.*

In addition, there is a unique partial function $\sigma_1 : \mathcal{H} \rightharpoonup |G|$ with domain $\delta(\Sigma)$ such that $|m| \circ \sigma = \sigma_1 \circ \delta$.

Proof. Clearly $\mathcal{H} = \tau_1(\Gamma) + \delta(|R|)$ with the restriction of $\tau_1 : \Gamma \to \tau_1(\Gamma)$ bijective. If $\delta(n) = \delta(n')$ for distinct nodes n and n' in R, then there is a chain from n to n' made of pieces like this one:

$$\begin{tikzcd}
& p \arrow[dl, mapsto, "\tau"'] \arrow[dr, mapsto, "|m|"] & & p' \arrow[dl, mapsto, "|m|"'] \arrow[dr, mapsto, "\tau"] & \\
\widetilde{n} & & p_1 & & \widetilde{n}'
\end{tikzcd}$$

with $\widetilde{n}, \widetilde{n}' \in |R|$, $p, p' \in |L|$, $p_1 \in |G|$, and it can be assumed that $\widetilde{n} \neq \widetilde{n}'$ and $p \neq p'$. Since m is a matching, $\widetilde{n}$ and $\widetilde{n}'$ are in Σ and $\sigma(\widetilde{n}) = \sigma(\widetilde{n}')$. The decomposition of $\mathcal{H}$ follows.
Now, let $n_1 \in \delta(\Sigma)$ and let us choose some $n \in \Sigma$ such that $n_1 = \delta(n)$. If σ_1 exists, then $\sigma_1(n_1) = \sigma_1(\delta(n)) = m(\sigma(n))$. On the other hand, if $n' \in \Sigma$ is another node such that $n_1 = \delta(n)$, then we have just proved that $\sigma(n) = \sigma(n')$, so that $m(\sigma(n))$ does not depend on the choice of n, it depends only on n_1. So, there is a unique $\sigma_1 : \mathcal{H} \rightharpoonup |G|$ as required, it is defined by $\sigma_1(n_1) = m(\sigma(n))$ for any $n \in \Sigma$ such that $n_1 = \delta(n)$.

Proposition 47 *Let $m : L \to G$ be a matching with respect to a rewrite rule $T = (L, R, \tau, \sigma)$. The pushout of τ and $|m|$ in* **Set***, with σ_1 as in proposition 46, underlies a cloning cone over T and m.*

Proof. First, let us define a graph H with set of nodes $\mathcal{H}$. According to proposition 46, and with the same notations, a graph H with set of nodes $\mathcal{H}$ is defined by imposing that τ_1 is strictly graphic on Γ, that δ is strictly graphic on Δ, and that each node $n_1 \in \delta(\Sigma)$ is a τ_1-clone of q_1, where $q_1 = \sigma_1(n_1)$.

Now, let us prove that δ underlies a graph morphism $d : R \to H$. Since δ is graphic on Δ, we have to prove that δ is also graphic on Σ. Let $n \in \Sigma$ and $n_1 = \delta(n)$. If n is unlabeled there is nothing to prove, otherwise let $q = \sigma(n)$, then q is labeled, $\mathcal{L}_R(n) = \mathcal{L}_L(q)$ and $\mathcal{S}_R(n) = \tau^*(\mathcal{S}_L(q))$. Then $m(q) = m(\sigma(n)) = \sigma_1(\delta(n)) = q_1$, and from the fact that m is a graph morphism we get $\mathcal{L}_L(q) = \mathcal{L}_G(q_1)$ and $|m|^*(\mathcal{S}_L(q)) = \mathcal{S}_G(q_1)$. The definition of H imposes $\mathcal{L}_G(q_1) = \mathcal{L}_H(n_1)$ and $\tau_1^*(\mathcal{S}_G(q_1)) = \mathcal{S}_H(n_1)$. Altogether, $\mathcal{L}_R(n) = \mathcal{L}_H(n_1)$ and $\mathcal{S}_H(n_1) = (\tau_1^*(|m|^*(\mathcal{S}_G(q))) = \delta^*(\tau^*(\mathcal{S}_G(q))) = \delta^*(\mathcal{S}_R(n))$, so that indeed δ is also graphic on Σ.

Finally, it is easy to check that this yields a cloning cone over T and m.

Theorem 48 *Given a rewrite rule $T = (L, R, \tau, \sigma)$ and a matching $m : L \to G$ with respect to T, the cloning cone over T and m defined in proposition 47 is a pushout of T and m.*

Proof. The cloning cone over T and m from proposition 47 is denoted $(m, d) : T \to T_1$ with $T_1 = (G, H, \tau_1, \sigma_1)$. Let us consider any cloning cone over T and m, say $(m, d') : T \to T_1'$ with $T_1' = (G', H', \tau_1', \sigma_1')$. Since (m, d) underlies a pushout of sets, there is a unique function $\eta : |H| \to |H'|$ such that $\eta \circ |d| = |d'|$ and $\eta \circ \tau_1 = \tau_1'$. Let $\Sigma = \mathrm{Dom}(\sigma)$ and $\Sigma_1 = \mathrm{Dom}(\sigma_1)$. Because the node functor is faithful, the result will follow if we can prove that $\eta(\Sigma_1) \subseteq \Sigma_1'$ and $\sigma_1' \circ \eta = \sigma_1$ on Σ_1, and that η underlies a graph morphism.
We have $\eta(\Sigma_1) = \eta(d(\Sigma)) = d'(\Sigma) \subseteq \Sigma_1'$, and for each $n_1 \in \Sigma_1$, let $n \in \Sigma$ such that $n_1 = d(n)$, then on one hand $\sigma_1'(\eta(n_1)) = \sigma_1'(\eta(d(n))) = \sigma_1'(d'(n)) = m(\sigma(n))$ and on the other hand $\sigma_1(n_1) = \sigma_1(d(n)) = m(\sigma(n))$, hence as required $\sigma_1'(\eta(n_1)) = \sigma_1(n_1)$.
In order to check that η underlies a graph morphism $h : H \to H'$, we use the

decomposition of $\mathcal{H}$ from proposition 46 and the construction of the cloning cone (m, d) in proposition 47. It follows immediately from lemma 24 that η is graphic on $\tau_1(\Gamma)$ and also on $d(\Delta)$. Let us prove that η is graphic on Σ_1. Let $n_1 \in \Sigma_1$, $q_1 = \sigma_1(n_1)$ and $n'_1 = \eta(n_1)$. Then $q_1 = \sigma'_1(n'_1)$ because $\sigma'_1 \circ \eta = \sigma_1$. So, n_1 is a τ_1-clone of q_1 and n'_1 is a τ'_1-clone of the same node q_1. This means that $\mathcal{L}_{H'}(n'_1) = \mathcal{L}_G(q_1) = \mathcal{L}_H(n_1)$ and that $\mathcal{S}_{H'}(n'_1) = (\tau'_1)^*(\mathcal{S}_G(q_1)) = \eta^*(\tau^*_1(\mathcal{S}_G(q_1))) = \eta^*(n_1)$. So, η is graphic on Σ_1, and since $d(\Sigma) \subseteq \Sigma_1$, it follows that η is graphic on $d(\Sigma)$. Altogether, η is graphic on the whole of $|H|$, which means that $\eta = |h|$ for a graph morphism $h : H \to H'$. This concludes the proof.

Definition 49 (Rewrite step) Given a rewrite rule $T = (L, R, \tau, \sigma)$ and a matching $m : L \to G$ with respect to T, the corresponding *rewrite step* builds the graph morphism $d : R \to H$, obtained from the cloning pushout of T and m.

Example 4. We go back to the rule $f(x) \to g(x, x)$ discussed at the end of section 3. This rule can be represented in our framework in different manners according to the way the term $g(x, x)$ is represented as a termgraph and also to the way the functions τ and σ are defined. We give below two different rules. Let G be the termgraph $1 : f(2 : a)$. The first rule (Rule1) rewrites the termgraph G into $1 : g(2 : a, 2)$, while the second rule (Rule2) rewrites G into $1 : g(2 : a, 3 : a)$. The node 2 and 3 in $1 : g(2 : a, 3 : a)$ are clones of node 2 in G.

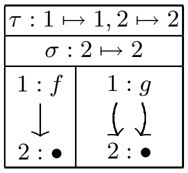


Fig. 5. Rule1

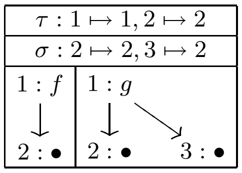
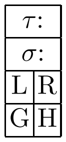

$\tau : 1 \mapsto 1, 2 \mapsto 2$
$\sigma : 2 \mapsto 2, 3 \mapsto 2$
$1 : f$
$2 : \bullet$
$1 : g$
$2 : \bullet$
$3 : \bullet$

Fig. 6. Rule2

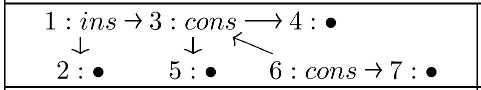
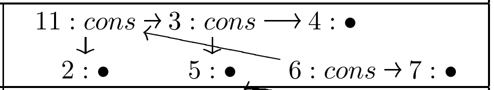
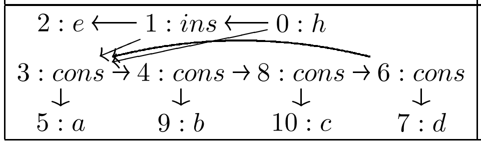
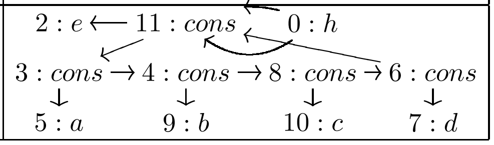
5 Examples

In this section, we give some illustrating examples. We represent a rewrite step $G \to H$ performed using a rewrite rule (L, R, τ, σ) as in the figure opposite. We assume in the given examples that the matching morphism $m : L \to G$ is such that $m(i) = i$.



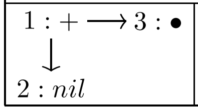
Insertion in a circular list

In this example we give a rule which defines the insertion of an element at the head of a circular list of size greater that one. In this rule, node 3 is the head of the list, and 6 is the last element of the list. Notice that, in R, the pointer to the head of the list, the second argument of node 6, has been moved from 3 to the new node 11 in R. The definition of τ is such that all pointers to the head of the list are moved from 3 to 11 ($\tau(3) = 11$). We apply the rule on a circular list of four items.

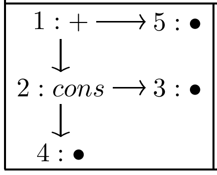
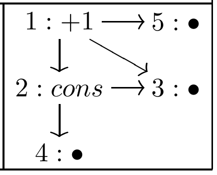
$\tau : 1 \mapsto 11, 3 \mapsto 11, i \mapsto i$ for $i \in \{2, 4, 5, 6, 7\}$	
$\sigma : 2 \mapsto 2, 4 \mapsto 4, 5 \mapsto 5, 7 \mapsto 7$	
	
	

Appending linked lists

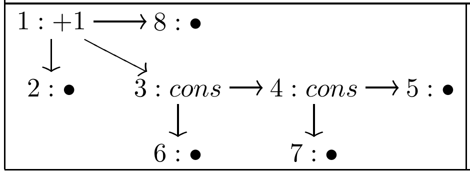
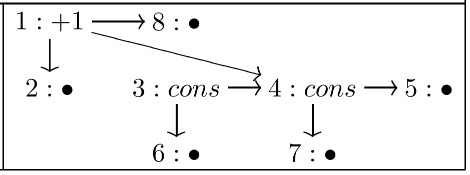
We now consider the rules for the operation "+" which appends two linked lists. The lists are supposed to be built with the constructors *cons*, and *nil*. The base case is defined when the first argument is *nil* as in the rule opposite.

$\tau : 1 \mapsto 3, 2 \mapsto 3, 3 \mapsto 3$	
$\sigma : 3 \mapsto 3$	
	$3 : \bullet$

When the first argument of + is a list different from *nil*, we call an auxiliary function denoted "+1", of arity 3. The role of this function is to go through the first list until the end and concatenate the two lists just by pointer redirection. The first call to the operation +1 is done by the rule opposite:

$\tau : i \mapsto i$ for $i \in \{1, 2, 3, 4, 5\}$	
$\sigma : 3 \mapsto 3, 4 \mapsto 4, 5 \mapsto 5$	
	

The second argument of +1 is used to go through the list starting at node 2 to get the last element of the list. This is implemented by the following rule :

$\tau : i \mapsto i$ for $i \in \{1, 2, 3, 4, 5, 6, 7, 8\}$	
$\sigma : 2 \mapsto 2, 6 \mapsto 6, 5 \mapsto 5, 7 \mapsto 7, 8 \mapsto 8$	
	

The last case for operation +1, is implemented as follows. We simply redirect the second edge from 3 to 4 (which is *nil*) towards 6 (e.g., $\tau(4) = 6$), which is the head of the second list to append. The overall result of the operation +1, is the head of first list, node $2 : \bullet$. This is implemented by $\tau(1) = 2$.

$\tau : 4 \mapsto 6, 1 \mapsto 2, i \mapsto i$ for $i \in \{2, 3, 5, 6\}$	
$\sigma : 2 \mapsto 2, 6 \mapsto 6, 5 \mapsto 5$	
$\begin{tikzcd} 1 : +1 \arrow[r] \arrow[d] \arrow[dr] & 6 : \bullet & \\ 2 : \bullet & 3 : cons \arrow[r] \arrow[d] & 4 : nil \\ & 5 : \bullet & \end{tikzcd}$	$\begin{tikzcd} 2 : \bullet & 3 : cons \arrow[r] \arrow[d] & 6 : \bullet \\ & 5 : \bullet & \end{tikzcd}$

Memory freeing

In this example we show how we can free the memory used by a circular list. As we are concerned with termgraphs where every function symbol has a fixed arity, it is not possible to create dangling pointers nor to remove useless pointers. This constraint is expressed by the fact that every node in a left-hand side L must have an image in the right-hand side R by τ.

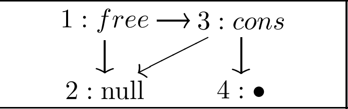
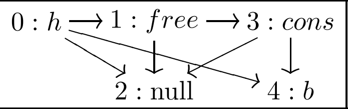
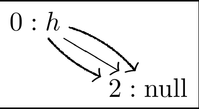
The operation $free$ has two arguments. The first one is a particular node labeled by a constant *null*. This constant is dedicated to be the target of the edges which were pointing the freed nodes. The second argument of $free$ is the list of cells to be freed.

Below, we give a rule defining the operation $free$ in the case of a list with at least two different elements. We also illustrate its application on a list of length two.

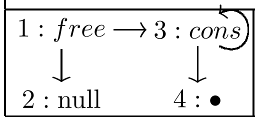
$\tau : 5 \mapsto 2, 3 \mapsto 2, i \mapsto i$ for $i \in \{1, 2, 4\}$	
$\sigma : 4 \mapsto 4$	
$\begin{tikzcd} 1 : free \arrow[r] \arrow[d] & 3 : cons \arrow[r] \arrow[d] & 4 : \bullet \\ 2 : \text{null} & 5 : \bullet & \end{tikzcd}$	$\begin{tikzcd} 1 : free \arrow[r] \arrow[d] & 4 : \bullet \\ 2 : \text{null} & \end{tikzcd}$
$\begin{tikzcd} 0 : h \arrow[r, Rightarrow] \arrow[drr] \arrow[drrr] & 1 : free \arrow[r] \arrow[d] & 3 : cons \arrow[r] \arrow[d] & 4 : cons \arrow[l, bend right] \arrow[d] \\ & 2 : \text{null} & 5 : a & 6 : b \end{tikzcd}$	$\begin{tikzcd} 0 : h \arrow[r] \arrow[dr] \arrow[drr] & 1 : free \arrow[r] \arrow[d] & 4 : cons \arrow[dl] \arrow[d] \\ & 2 : \text{null} & 6 : b \end{tikzcd}$

Notice that pointers incident to nodes 3 and 5 are redirected towards 2.

There are two cases for lists with one element. The following rule specifies the case where the last element of the list is obtained after freeing other elements of the list. We illustrate the rewrite rule on the graph obtained earlier (up to renaming of nodes).

$\tau : i \mapsto 2$ for $i \in \{1, 2, 3, 4\}$	
$\sigma :$	
	2 : null
	

Finally, because of the injectivity condition on matching, we have to consider the special case of lists of size one. This is done by the following rule:

$\tau : i \mapsto 2$ for $i \in \{1, 2, 3, 4\}$	
$\sigma :$	
	2 : null

6 Conclusion

We have proposed a new way to define termgraph rewrite rules. Rules are quite simple. A rule is a tuple (L, R, τ, σ) where L and R are termgraphs representing the left-hand and the right-hand sides of the rule, τ is a mapping from the nodes of L to those of R and σ is a partial function from nodes of R to nodes of L. τ describes how incident edges of the nodes in L are connected in R. It should be noted that τ is not required to be a graph morphism as in the classical algebraic graph transformation approaches [4, 6]. As for σ, it is useful only when one needs to clone some parts of L. We defined rewrite steps as pushouts in an appropriate category as shown in section 4.

The proposed rewrite systems offer the possibility to transform cyclic termgraphs either by performing local edge redirections or global edge redirections, as defined in [5] following a DPO approach, but provides also new features not present in [5] such as cloning or deletion of nodes.

Besides the algorithmic approaches to termgraph transformation (e.g. [2]), a categorical framework dedicated to *cyclic* termgraph transformation could be found in [3] where the authors propose, following [12], a 2-categorical presentation of termgraph rewriting. They almost succeeded to represent the full operational view of termgraph rewriting as defined in [2], but differ on rewriting circular redexes. For example, the application of the rewrite rule $f(x) \to x$ on the termgraph $n : f(n)$ yields the same termgraph (i.e. $n : f(n)$) according to [2] but yields an unlabeled node, say $p : \bullet$, according to [3]. The definition of rewrite rules that we propose in this paper allows us to make a clear distinction between the two behaviours. The rule $(n : f(m : \bullet), p : \bullet, \tau = \{n \mapsto p, m \mapsto p\}, \sigma = \{\})$

behaves as in [3] when applied on $n : f(n)$, whereas the behaviour described in [2] can be obtained by simply declaring that node p is a clone of node m via σ as in the following rule $(n : f(m : \bullet), p : \bullet, \tau = \{n \mapsto p, m \mapsto p\}, \sigma = \{p \mapsto m\})$.

Future works include the generalization of the proposed systems to other graphs less constrained than termgraphs. This would allow us to require from τ, in a rule (L, R, τ, σ), to be a partial function like in the single pushout approach [6].

References

1. Z. Ariola and J. Klop. Equational term graph rewriting. *Fundamenta Informaticae*, 26(3-4), 1996.
2. H. Barendregt, M. van Eekelen, J. Glauert, R. Kenneway, M. J. Plasmeijer, and M. Sleep. Term graph rewriting. In *PARLE'87*, pages 141–158. Springer Verlag LNCS 259, 1987.
3. A. Corradini and F. Gadducci. A 2-categorical presentation of term graph rewriting. In *7th International Conference on Category Theory and Computer Science (CTCS 97)*, volume 1290 of *Lecture Notes in Computer Science*, pages 87–105. Springer, 1997.
4. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation - part I: Basic concepts and double pushout approach. In *Handbook of Graph Grammars*, pages 163–246, 1997.
5. D. Duval, R. Echahed, and F. Prost. Modeling pointer redirection as cyclic termgraph rewriting. *Electr. Notes Theor. Comput. Sci.*, 176(1):65–84, 2007.
6. H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. Algebraic approaches to graph transformation - part ii: Single pushout approach and comparison with double pushout approach. In *Handbook of Graph Grammars*, pages 247–312, 1997.
7. H. Ehrig, M. Pfender, and H. J. Schneider. Graph-grammars: An algebraic approach. In *14th Annual Symposium on Foundations of Computer Science (FOCS), 15-17 October 1973, The University of Iowa, USA*, pages 167–180. IEEE, 1973.
8. R. Kennaway. On "on graph rewritings". *Theor. Comput. Sci.*, 52:37–58, 1987.
9. M. Löwe. Algebraic approach to single-pushout graph transformation. *Theor. Comput. Sci.*, 109(1&2):181–224, 1993.
10. S. Mac Lane. *Categories for the Working Mathematician*, volume 5. Springer-Verlag, second edition edition, 1998.
11. D. Plump. Term graph rewriting. In H. Ehrig, G. Engels, H. J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, pages 3–61. World Scientific, 1999.
12. A. J. Power. An abstract formulation for rewrite systems. In *Category Theory and Computer Science*, volume 389 of *Lecture Notes in Computer Science*, pages 300–312. Springer, 1989.
13. J. C. Raoult. On graph rewriting. *Theoretical Computer Science*, 32:1–24, 1984.
14. M. R. Sleep, M. J. Plasmeijer, and M. C. J. D. van Eekelen, editors. *Term Graph Rewriting. Theory and Practice.* J. Wiley & Sons, Chichester, UK, 1993.